Atom

Atom is the most commonly used IDE on the CMR team.

Installation and Setup

Follow the instructions here and here.

Using Atom

Proto-Repl

After following the installation and setup instructions, you'll have proto-repl installed and configured in Atom. Proto-repl is a dev environment and REPL for Atom.

The website has a lot of good information on how to use proto-repl, especially the key bindings. There's more on the keybindings here.

To get started with proto-repl, clone the proto-repl-demo project and go through the demos.md file. This allows you to experiment with the features and key bindings of proto-repl.

Start a REPL

To open CMR in atom, in a terminal navigate to <home-dire>/workspaces/cmr and run the command: 'atom .'

To open an individual library or microservice, navigate to that folder and run 'atom .' (i.e. workspace/cmr/search-app to open just the search app.)

To start a REPL, open the project.clj file for dev-system or the library/microservice you opened. To open a REPL for all of CMR, you must open and click in the project.clj for dev-system. Press Cmd-Alt-Shift-L to start a new REPL with your cursor in the project.clj

Once the REPL is opened, you can execute commands in the REPL or in the code.

Run Tests

To run all unit/integration tests, there is a run all unit tests button in Atom. When the tests are complete, the REPL will show the number of failures and errors.

To run an individual test, place the cursor on the test and run the command: Cmd-Alt-t This will run the fixtures (setup) as well as the tests. The output in the REPL will be nil if the test passes, otherwise errors will be displayed.

To run all tests in a file, run the command: Cmd-Alt-x

To run all tests in a microservice/library, open the microservice/library in Atom and run all tests.

Debugging

Proto-repl comes with some debugging features. Read the "Saving and Viewing Local Binding Values" on the proto-repl page for how to use proto-repl debugging. Proto-repl can save a function's local values so they can be displayed later and even defined as a global variable.

Another debugging trick is to assign variables to global defines in the function so the defines can be used after the code has been executed. You can see the value and operate on it, making debugging quicker. You can run just the problem block of code multiple times using the defined value, rather than having to run the entire test.

When working in a file, you can put a (comment ...) block in to keep your debug code and try different things within that namespace.

Reloading Code

The CMR systems are setup so that the system can be quickly shutdown, code reloaded off the file system, and then restarted any time a

developer wants. cmd-alt-r will run invoke the user/reset function to do this.

cmd-alt-r will only reload the files that have changed. Occasionally a more thorough refresh is required. Use cmd-alt-shift-r to reload every file. This is called a "super refresh"

Troubleshooting Common Issues

Errors during refresh that look like they were not caused by code you are working on can usually be solved by doing a super refresh.

If you've recently gotten latest code and are seeing errors refreshing or unit tests failing, you may need to run 'lein modules do clean, install, clean' in your terminal in the root directory of CMR. Dependencies may have changed making this necessary. Make sure to close and restart your REPL after doing this.

If you're trying to refresh or run a test and you're getting a "namespace not found" error, go to that file and press cmd-, then f to reload the file. Restarting the REPL will also work in this case but is more time consuming.